Universidad Nacional de Asunción
Lic. en Ciencias Informáticas
Facultad Politécnica

Árboles Binarios y Árboles Binarios de Búsqueda

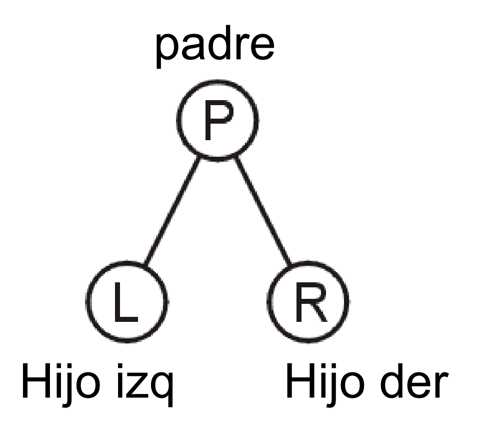
Prof. Ing. Derlis Zárate
ProfDerlisZarate@gmail.com

Problemas con Árboles Grales.

- Problema con los árboles generales
  - Algunas operaciones son $\mathbf{O}(n)$ donde n es el número de hijos
  - No hay un orden natural entre la raiz y sus hijos

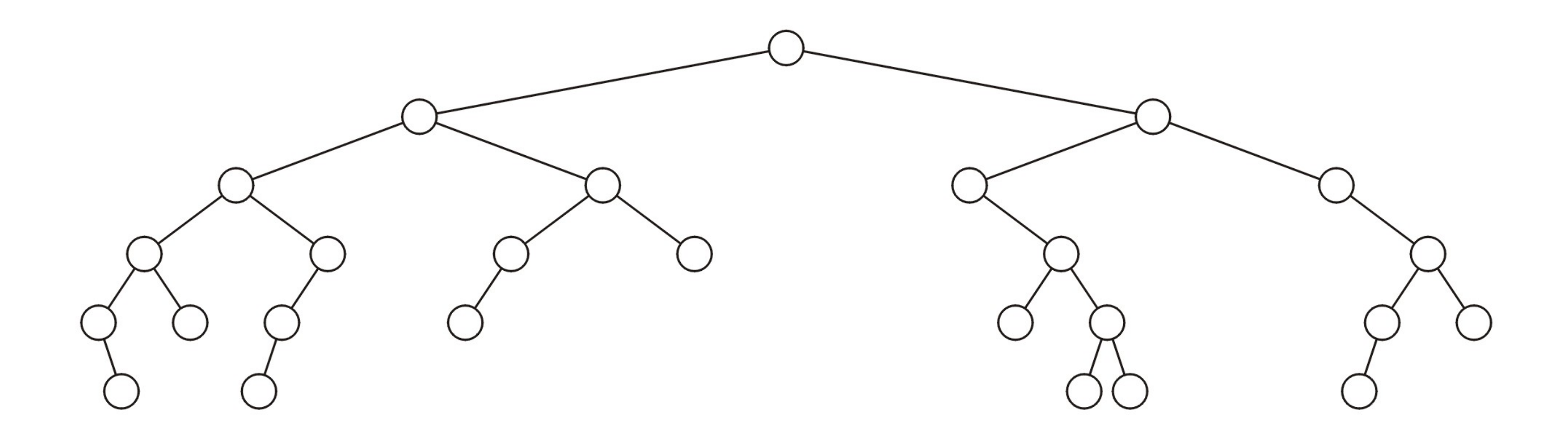
Árboles Binarios

- Consideremos un árbol general pero con la restricción de que cada nodo solo pueda tener 2 hijos.
  - Cada nodo es vacío o es otro árbol binario.
  - Esta restricción nos permite referirnos a los hijos simplemente como *izquierdo* y *derecho.*



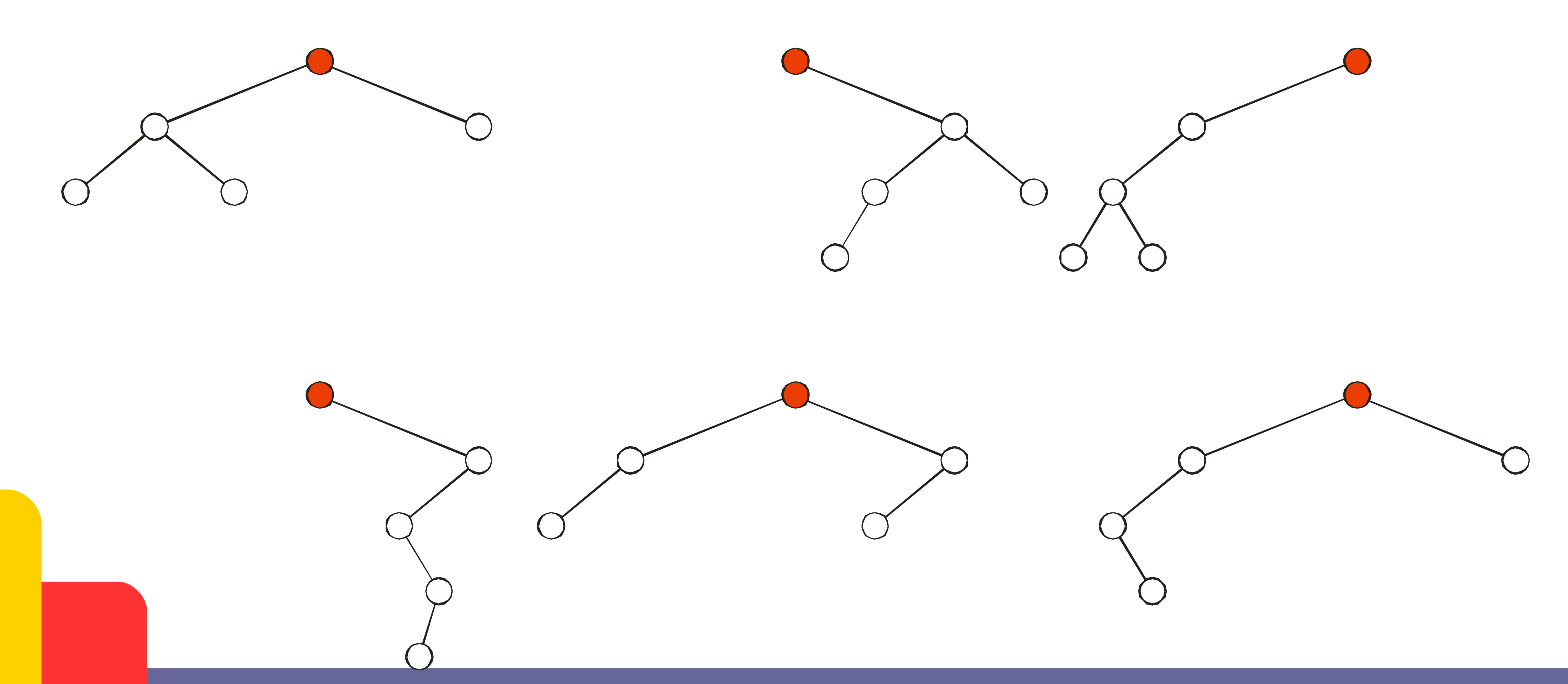
Árboles Binarios

- Esto nos permite diagramar un árbol binario con el siguiente patrón general:



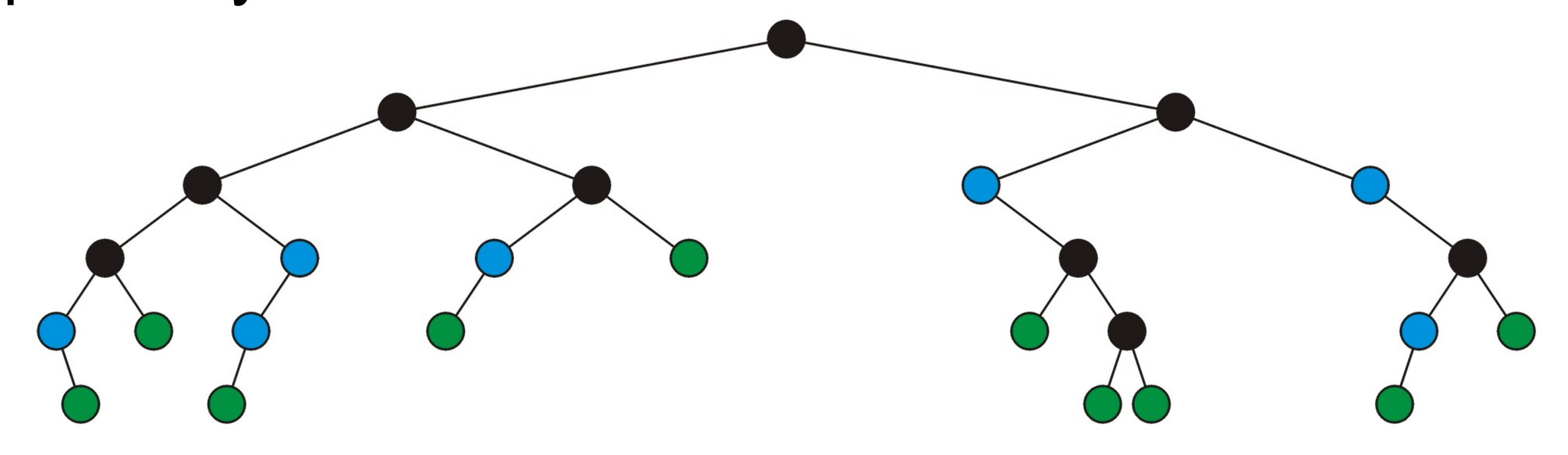
Árboles Binarios

- Los siguientes árboles son variaciones de árboles binarios de 5 nodos:



Árboles Binarios

- Un *nodo lleno* es un nodo donde tanto los subárboles izquierdo y derecho no son vacíos:

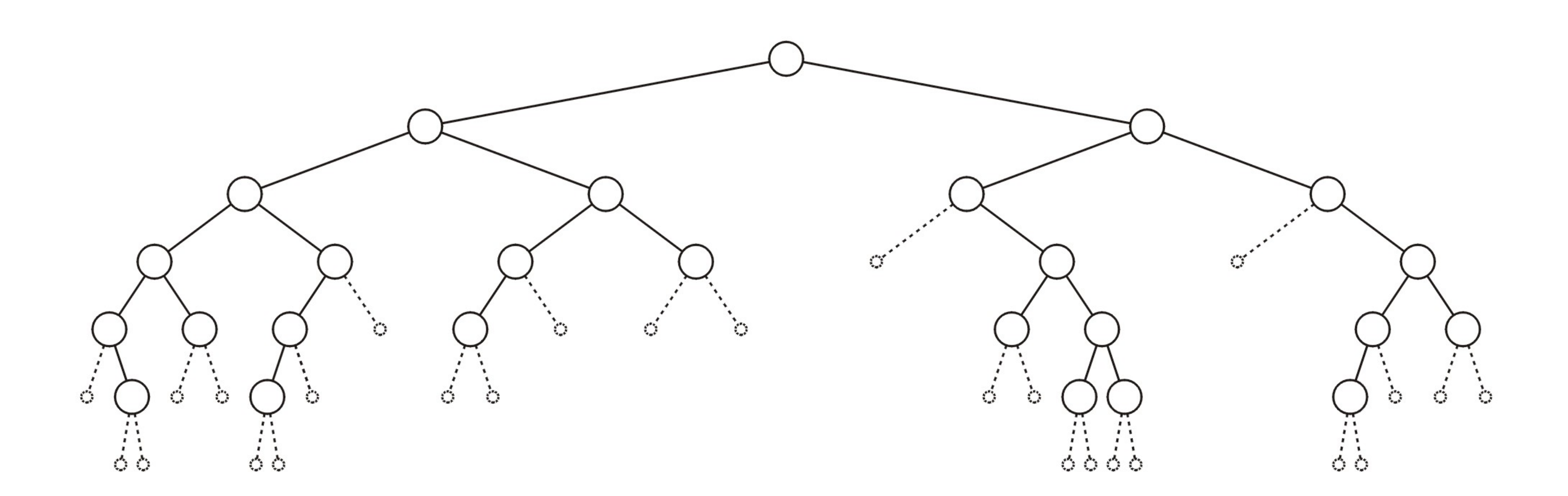


- Referencia:

nodos llenos **nodos no llenos** **nodos hoja**

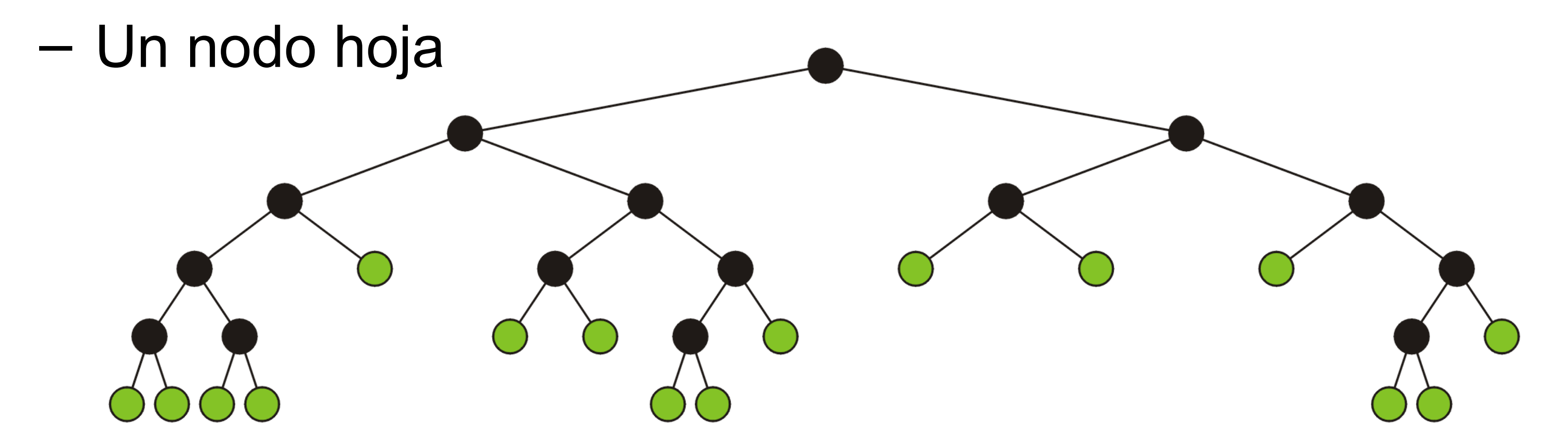
Árboles Binarios

- Un *nodo vacío* o *subárbol nulo* es cualquier lugar del árbol donde se puede insertar un nuevo nodo hoja:



Árboles Binarios

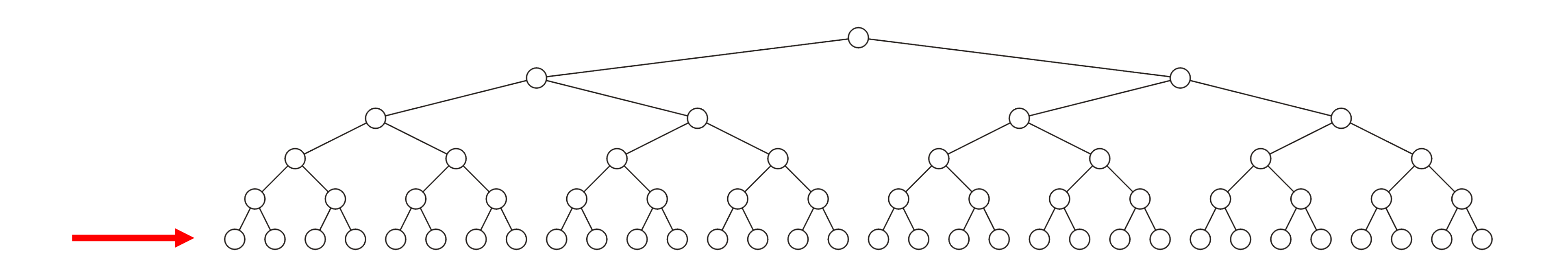
- Un *árbol binario lleno* es un árbol en el que cada nodo es:
  - Un nodo lleno o
  - Un nodo hoja



- Esto tiene aplicaciones en
  - Árboles de Expresiones
  - Codificación Huffman (compresión de textos)

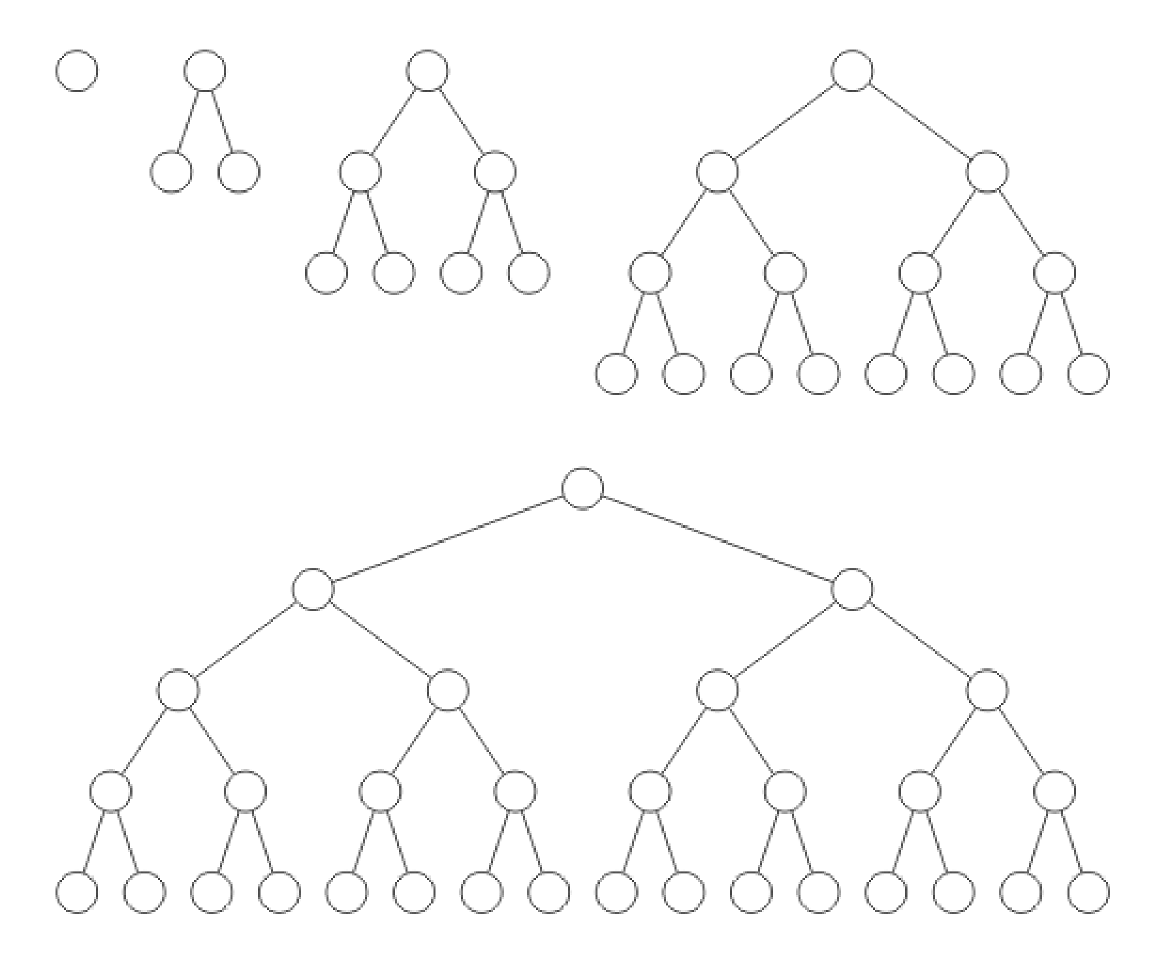
Árboles Binarios

- Árboles Binarios Perfectos:
  – Un árbol binario perfecto de altura *h* es un árbol binario donde
    - Todas las hojas tienen la misma profundidad h
    - Todos los otros nodos son llenos



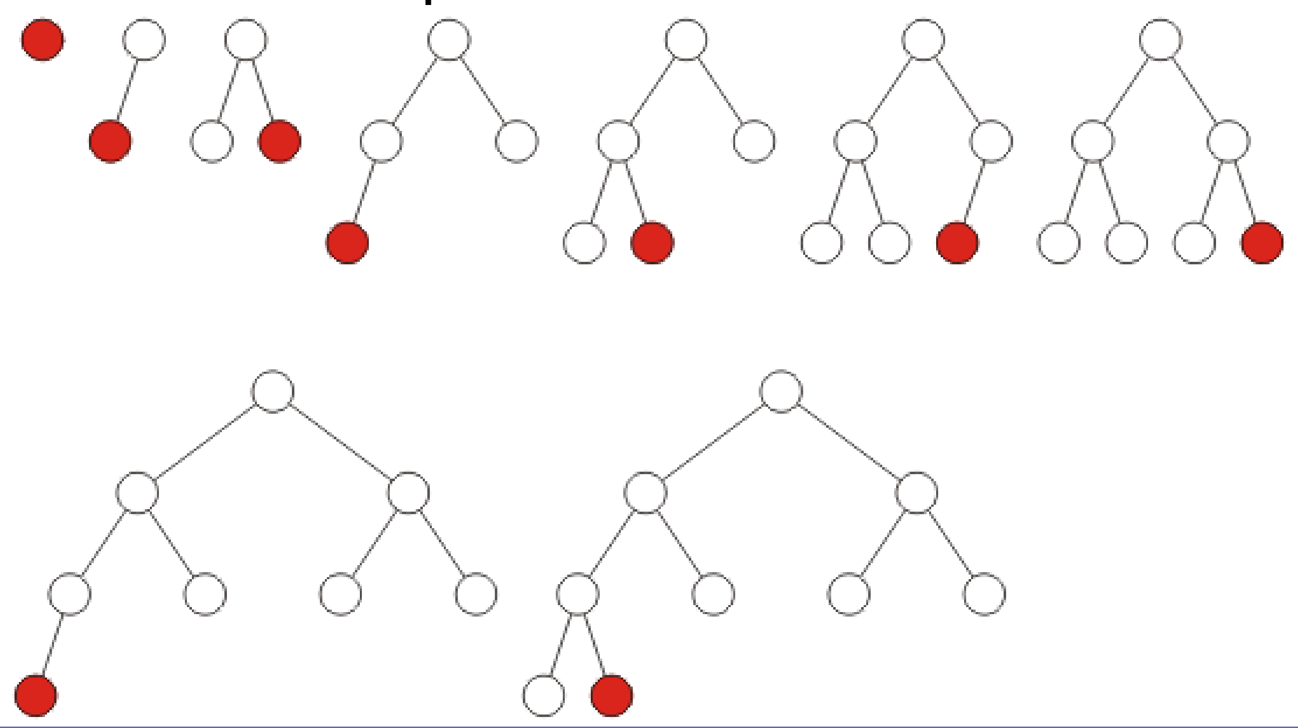
Árboles Binarios

- Ejemplos de árboles binarios perfectos de altura h=0,1,2,3 y 4



Árboles Binarios

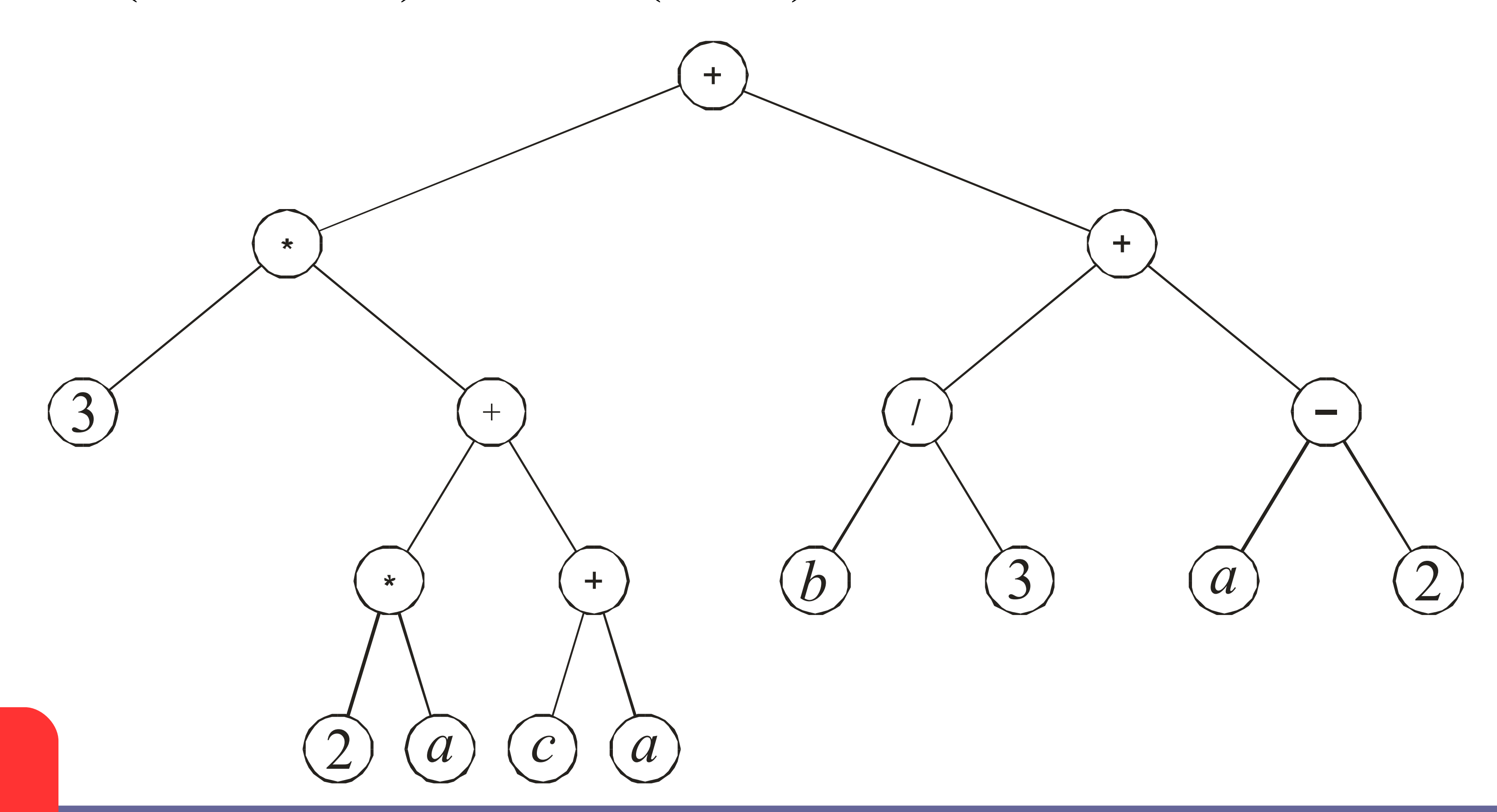
- Un árbol binario es *completo* si está rellenado en cada nivel de izquierda a derecha.
- En un árbol completo de altura h, todos los niveles excepto posiblemente el nivel h-1 están llenos. El último nivel tiene todos sus nodos de izquierda a derecha



Ejemplo aplicación

- Se puede usar un árbol binario para implementar un árbol de expresiones:

$$3(2a + c + a) + b/3 + (a - 2)$$

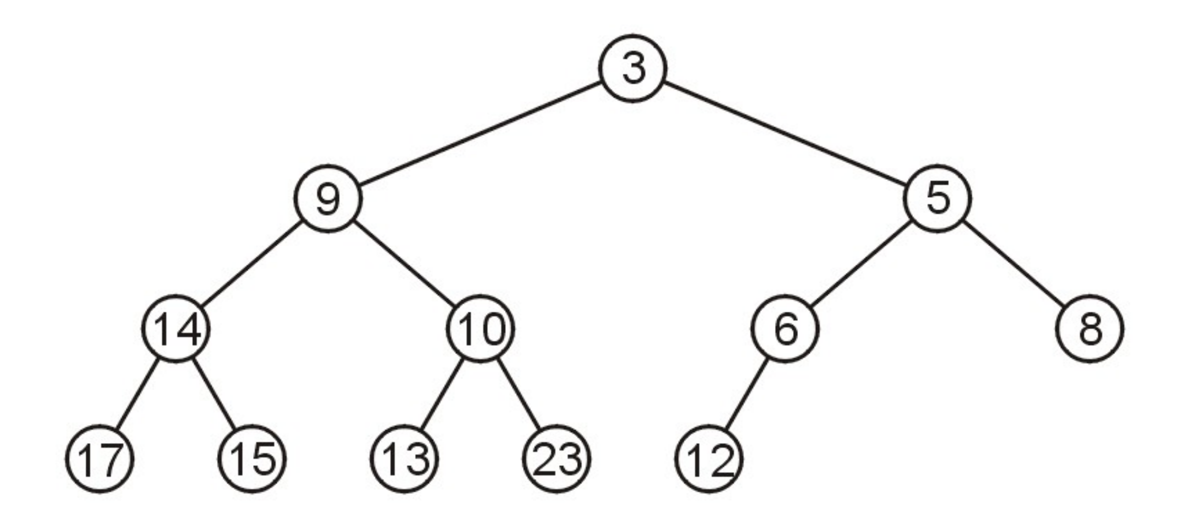


Implementación con arrays

- Al ser binario el árbol, podemos almacenar el árbol completo como un array.
- Simplemente recorremos el array en amplitud, ubicando las entradas en el array.

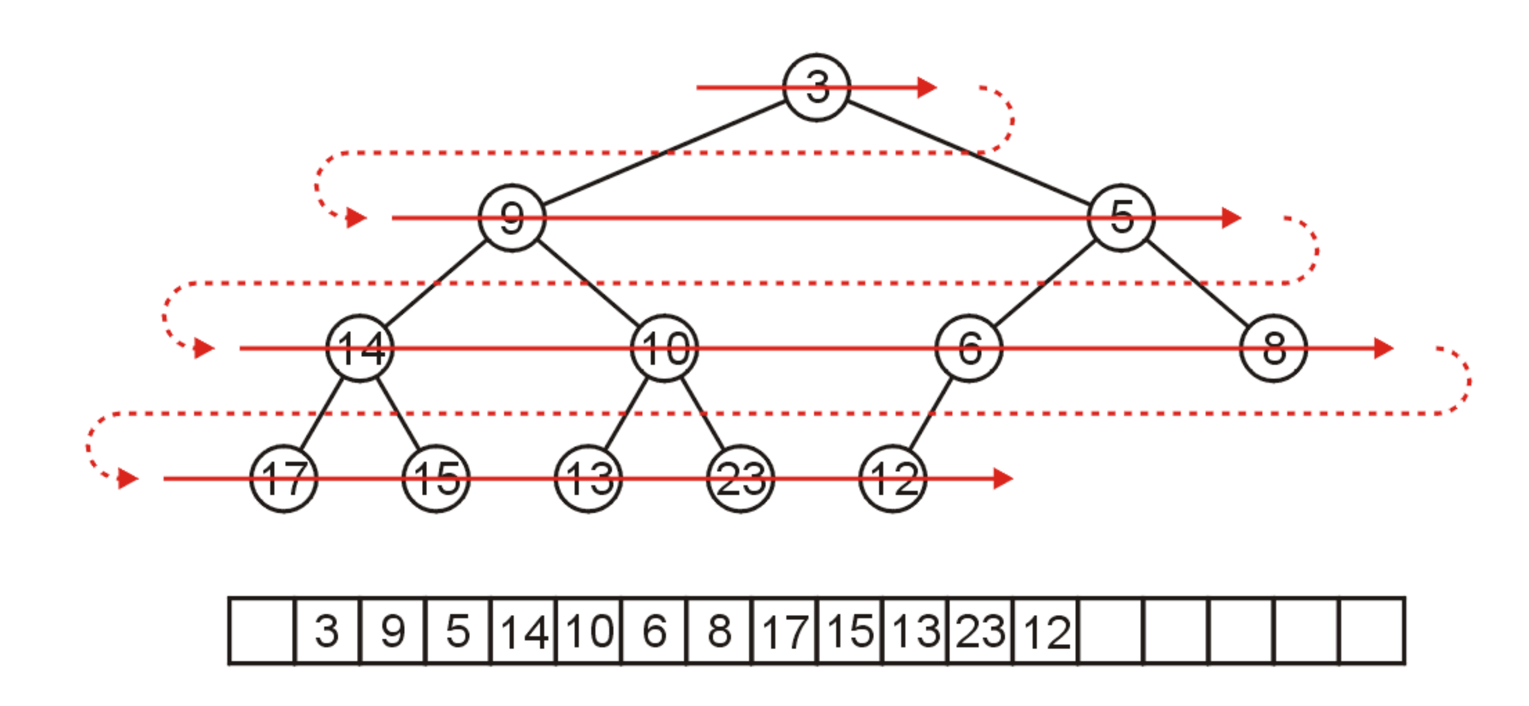
Implementación con arrays

- Por ejemplo, para el siguiente árbol:



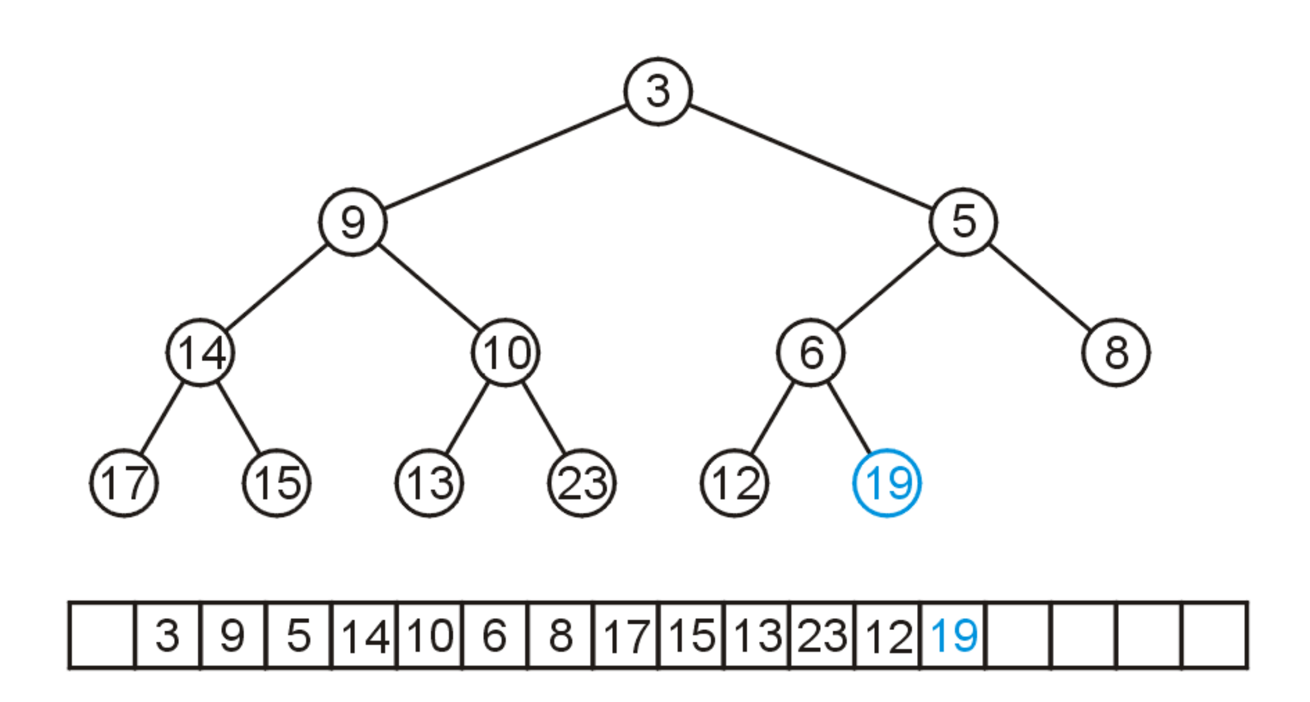
Implementación con arrays

- Lo recorremos en amplitud y rellenamos el array:



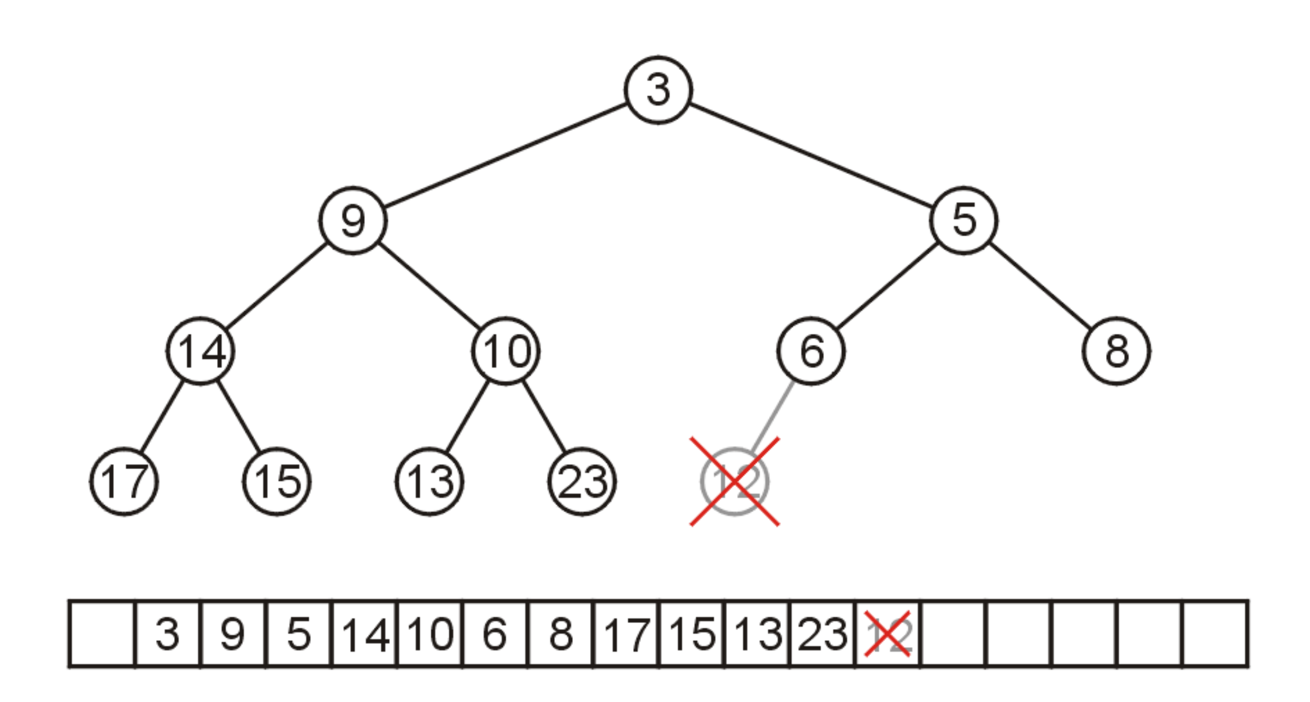
Implementación con arrays

- Para insertar otro nodo y mantener la estructura de árbol binario completo, insertamos en el siguiente lugar del array



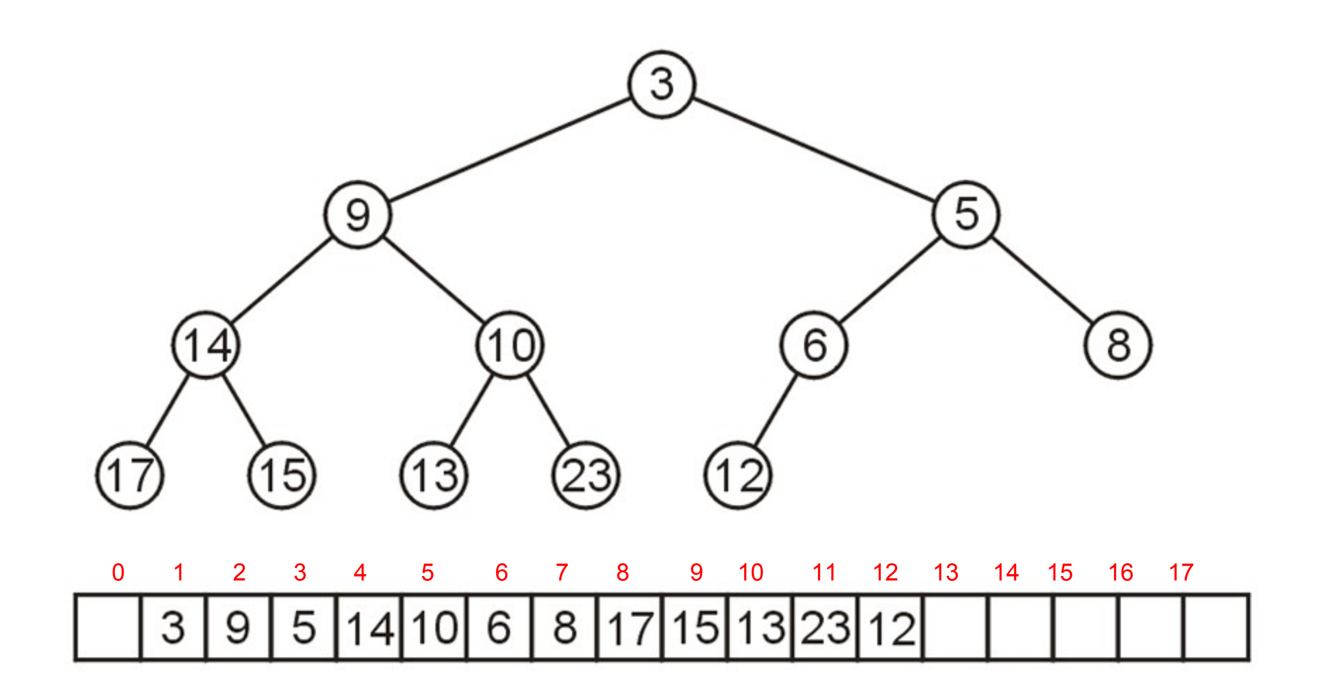
Implementación con arrays

- Para eliminar un nodo y mantener la estructura de árbol completo, debemos eliminar el último elemento del array



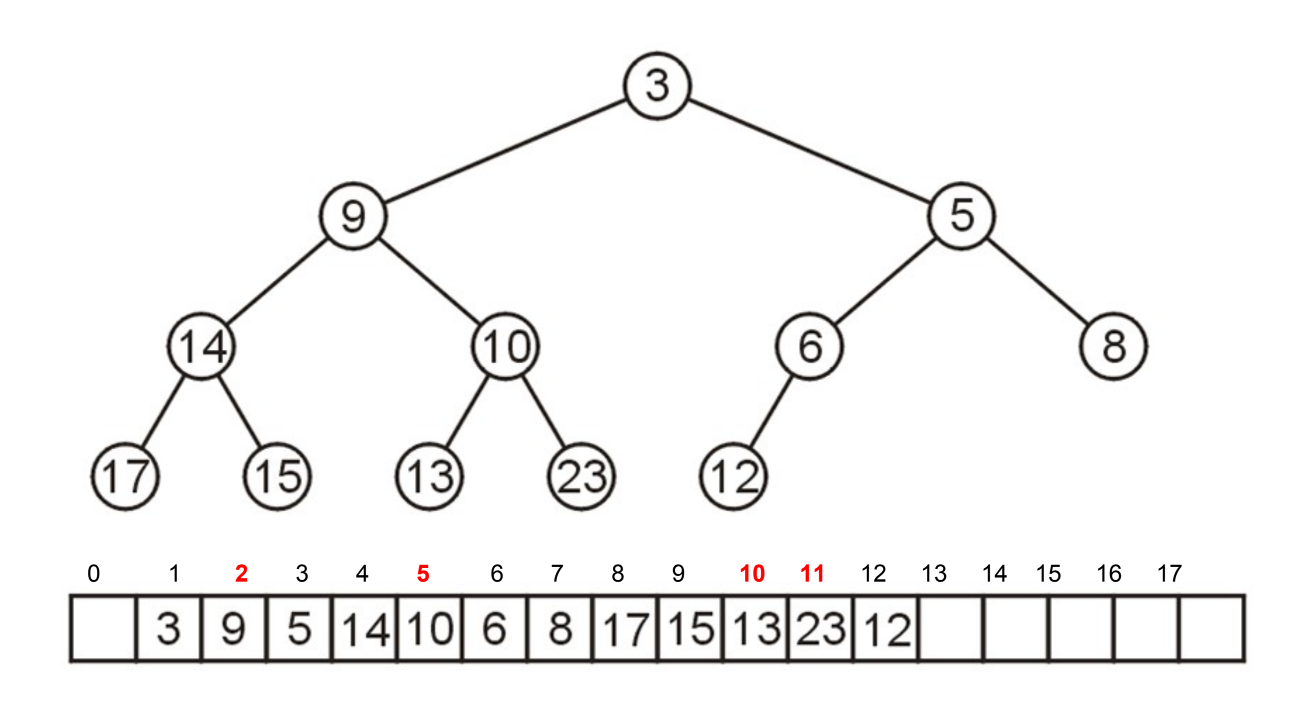
Implementación con arrays

- Se deja el primer espacio en blanco, para facilitar el cálculo:
  - Los hijos del nodo con índice *k* están en las posiciones *2k* y *2k+1*
  - El padre del nodo con índice *k* está en $k \div 2$



Implementación con arrays

- Por ejemplo, el nodo 10 tiene índice 5:
  - Sus hijos 13 y 23 tienen índices 10 y 11, respectivamente
  - Su padre es el nodo 9, con índice 2

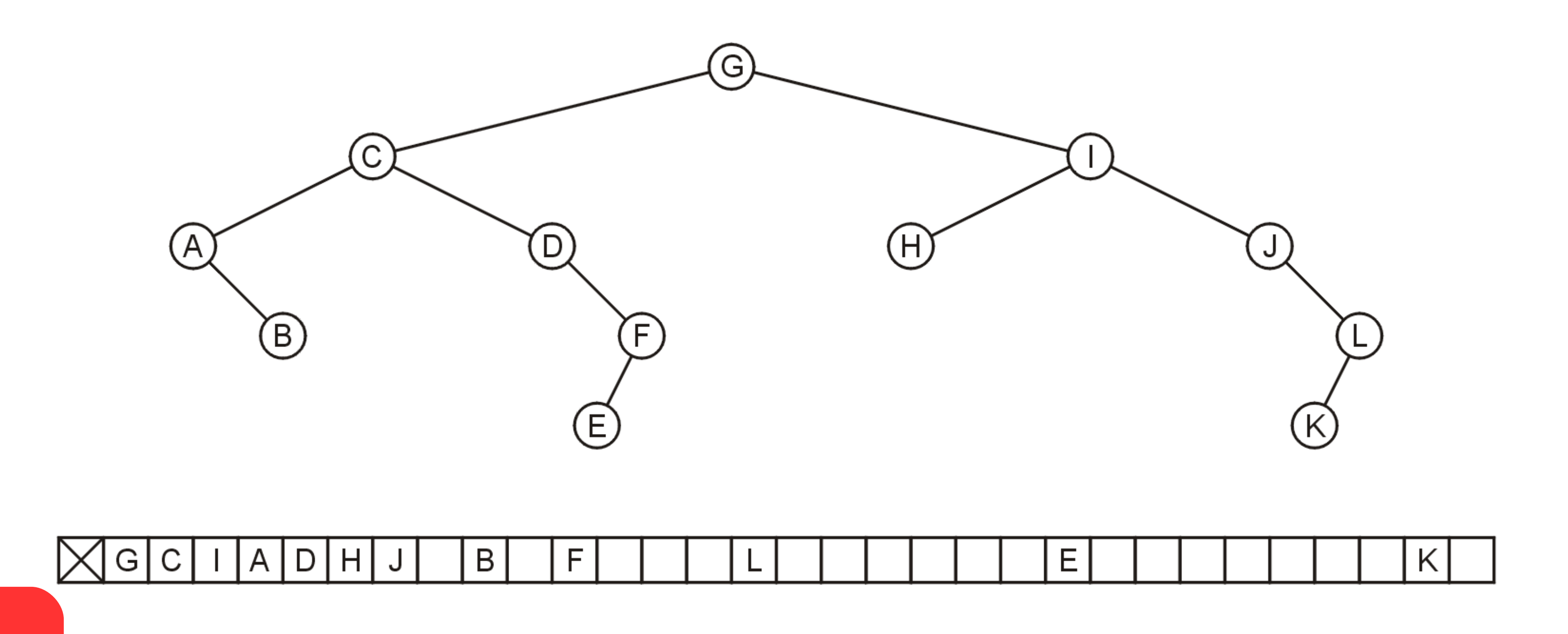


Implementación con arrays

- Pregunta existencial: porqué no almacenar entonces cualquier árbol como array?
- La justificación esencial:
  - Hay un potencial significativo y alta probabilidad de una gran pérdida/desperdicio de memoria

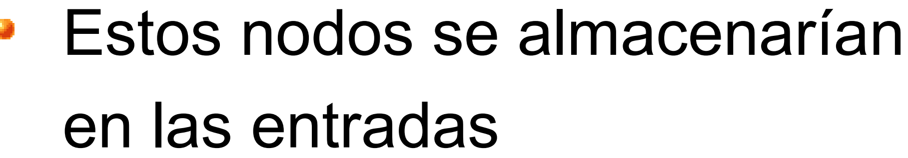
Implementación con arrays

- Consideremos el árbol con 12 nodos siguiente, almacenado como array.
  - Requiere un array de tamaño 32
  - Agregar un hijo al nodo K, dobla la memoria requerida.



Implementación con arrays

- En el peor caso, la cantidad de memoria requerida es exponencial.



- Estos nodos se almacenarían en las entradas

  1, 3, 6, 13, 26, 52, 105

A G B F E C D

Implementación con referencias

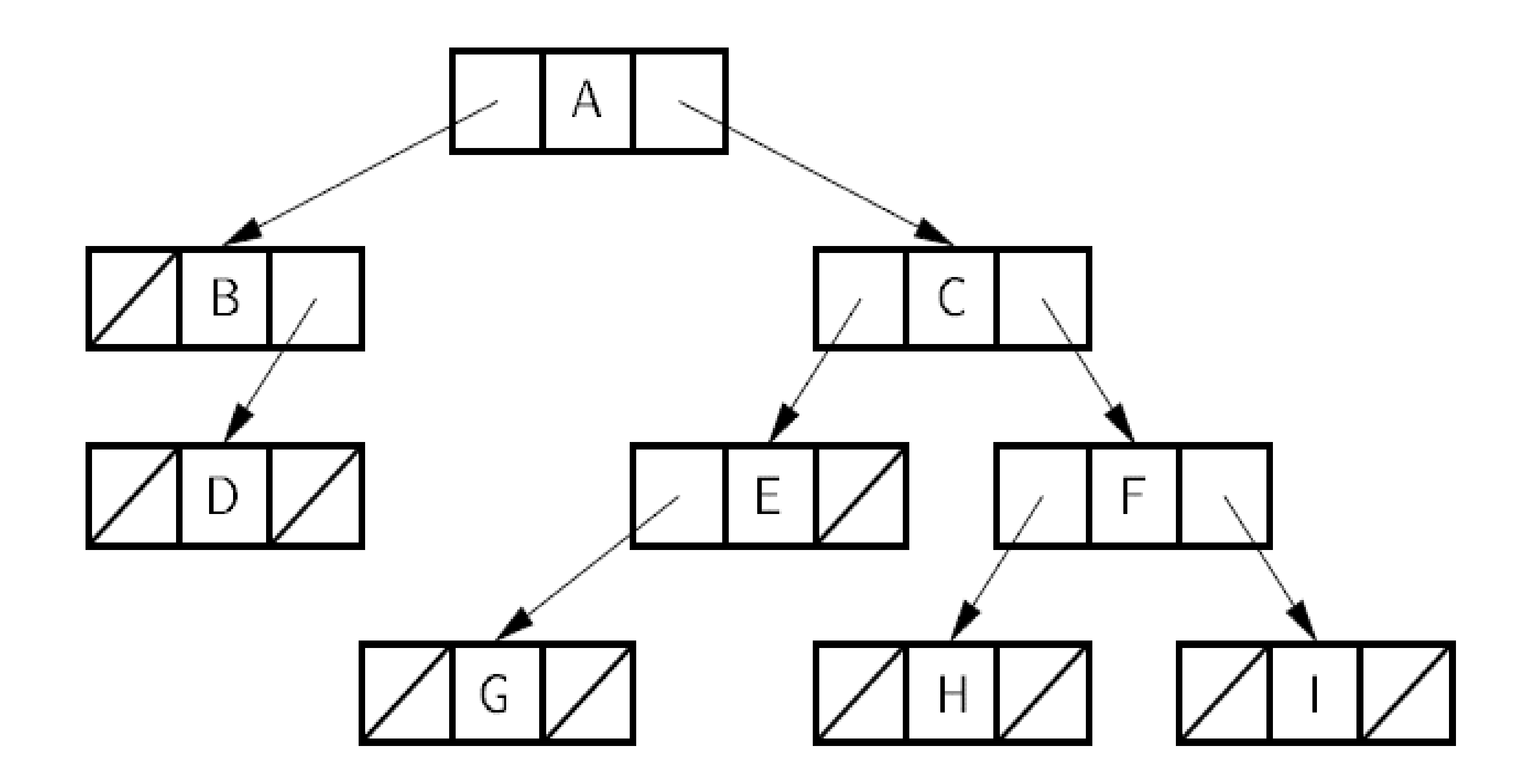
```
public class NodoBinario<TipoDeDato extends Comparable<TipoDeDato>> {
    Private TipoDeDato dato;
    Private NodoBinario<TipoDeDato> izq;
    Private NodoBinario<TipoDeDato> der;

    … setters y getters …

    //constructor
    public NodoBinario(TipoDeDato valor, NodoBinario hIzq,
                       NodoBinario hDer) {
        This.dato = valor;
        This.izq = hIzq;
        This.der = hDer;
    }
};
```

Implementación con referencias

- Ejemplo de cómo se vería internamente las referencias en memoria:



Árboles Binarios de Búsqueda

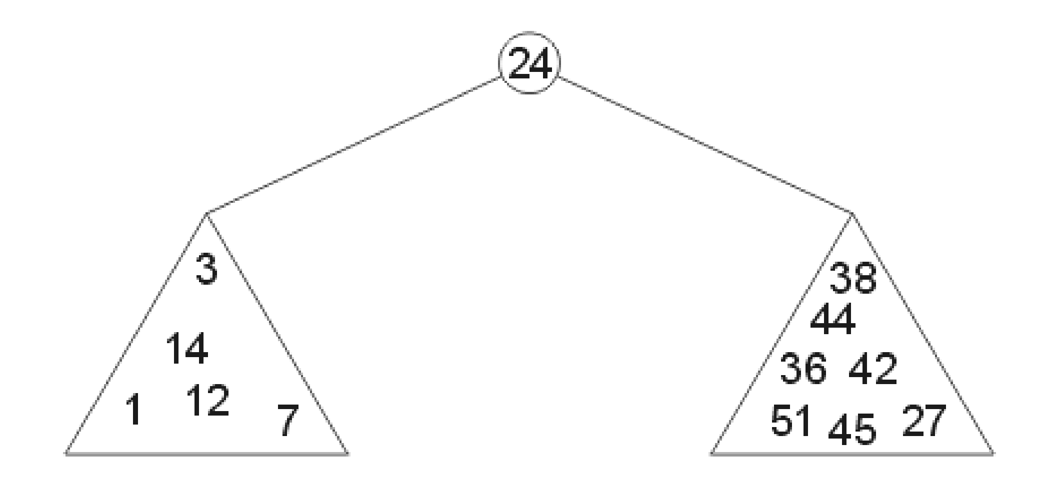
- Hasta ahora estudiamos varias estructuras de datos para almacenar datos y también para ordenar datos.
- Algunas de ellas tenian mejor comportamiento en las inserciones, pero no en las búsquedas.
- Otras sin embargo funcionaban mejor para accesos a elementos que para las inserciones.

Árboles Binarios de Búsqueda

- Todas las operaciones en un árbol binario perfecto son $\mathbf{O}(\log(n))$, entonces, podemos usar árboles binarios para almacenar información?
- Dado el nodo raíz, tenemos 2 subárboles, el izquierdo y el derecho
  - Supongamos que todos los datos en el subárbol izquierdo son menores que la raíz, y
  - Supongamos que todos los datos del subárbol derecho son mayores que la raíz.

Árboles Binarios de Búsqueda

- Gráficamente vemos algo parecido a ésto:



Árboles Binarios de Búsqueda

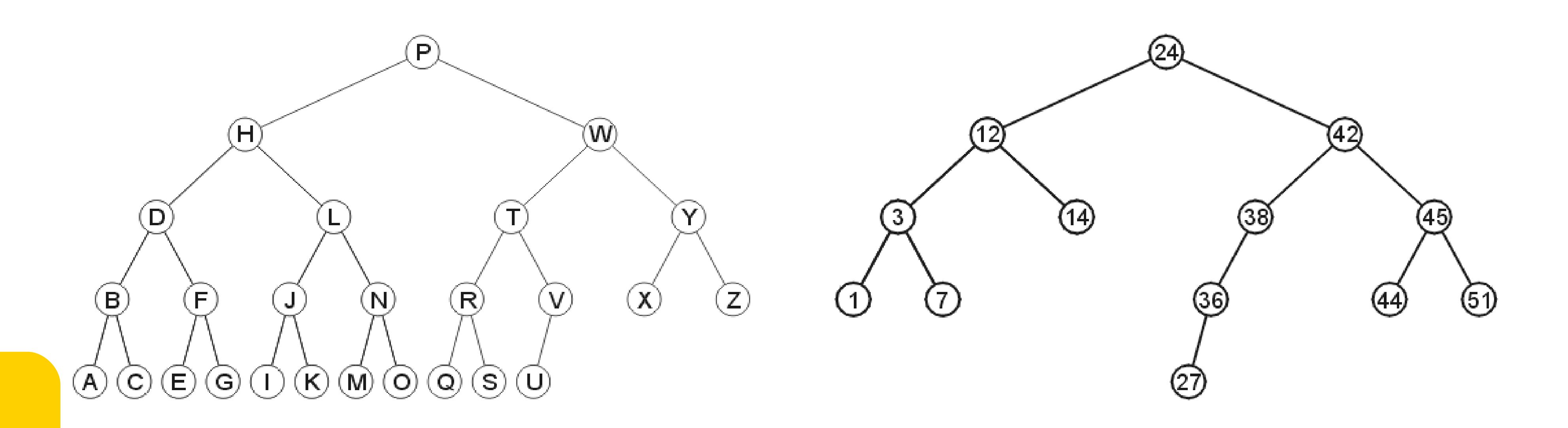
- En este caso, si estamos buscando un elemento en el árbol, podemos comparar el elemento buscado con la raíz y:
  - Si el elemento raíz es igual a lo que buscamos, terminamos la búsqueda
  - Si el elemento buscado es menor que lo que está en la raíz, continuamos la búsqueda en el subárbol izquierdo.
  - De lo contrario, buscamos en el subárbol derecho.

Árboles Binarios de Búsqueda

- Con todo ésto, podemos definir a un árbol binario de búsqueda como el árbol que cumple las siguientes propiedades:
  - Es un árbol binario
  - El subárbol izquierdo (si lo hay) es un árbol binario de búsqueda con todos los elementos menores que la raiz.
  - El subárbol derecho (si lo hay) es un árbol binario de búsqueda con todos los elementos mayores que la raiz.

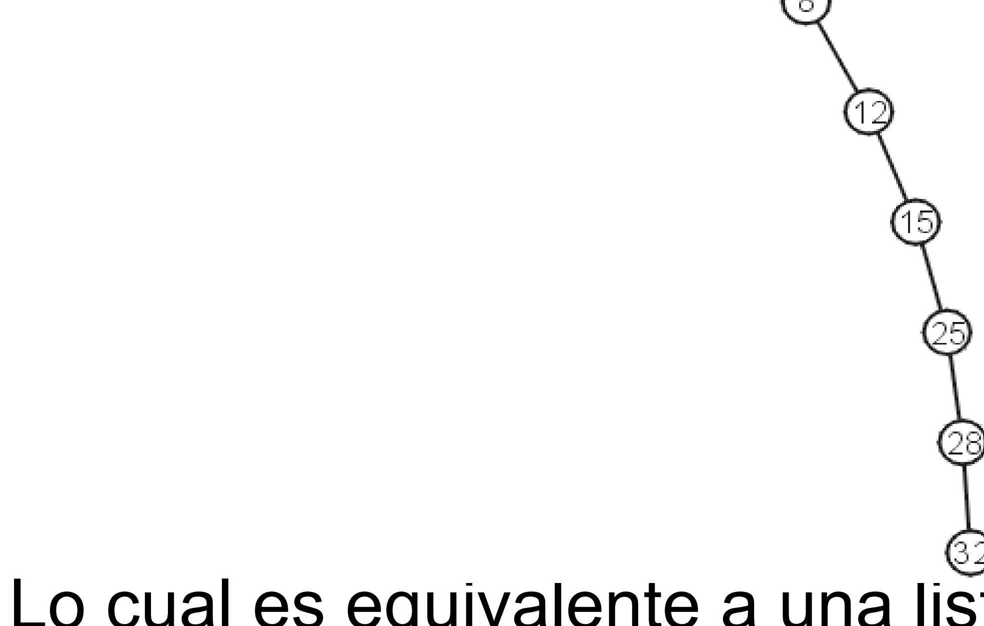
Árboles Binarios de Búsqueda

- A continuación se ven ejemplos de árboles binarios de búsqueda completos, y árboles binarios de búsqueda cercanos a ser completos (a veces conocidos como árboles balanceados)



Árboles Binarios de Búsqueda

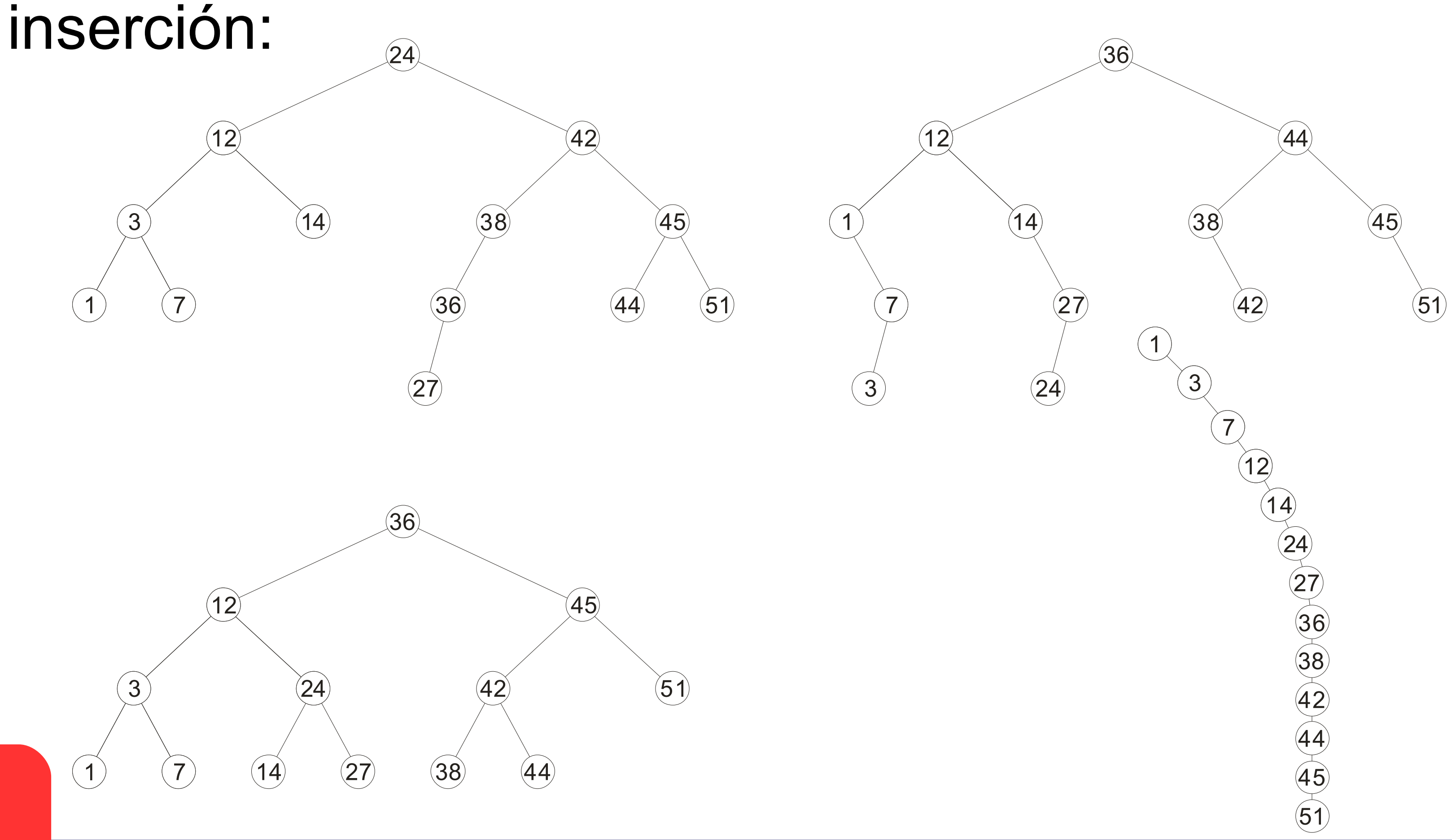
- Lamentablemente, esto también es un árbol binario de búsqueda:



- Lo cual es equivalente a una lista enlazada $\mathbf{O}(n)$

Árboles Binarios de Búsqueda

- Pueden haber varias representaciones diferentes para el mismo conjunto de datos. Depende del orden de inserción:

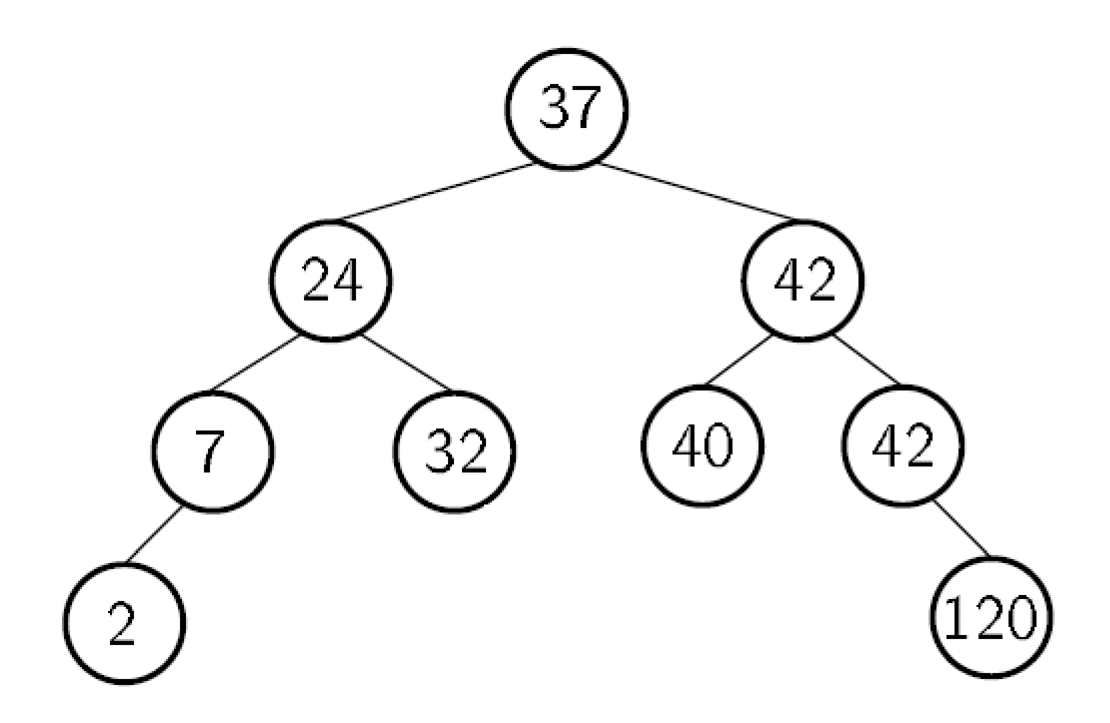


Elementos duplicados

- Al igual que con los árboles generales que estudiamos, aquí asumimos que en cualquiera de nuestros árboles binarios de búsqueda no hay elementos duplicados.
- Se pueden considerar elementos duplicados, con modificaciones a los algoritmos que estudiaremos.

Operaciones elementales

- Buscar el 32
- Buscar el 40
- Buscar el 120
- Insertar el 35
- Insertar el 41
- Insertar el 6
- Eliminar el 120
- Eliminar el 24
- Eliminar la raiz



Operaciones elementales

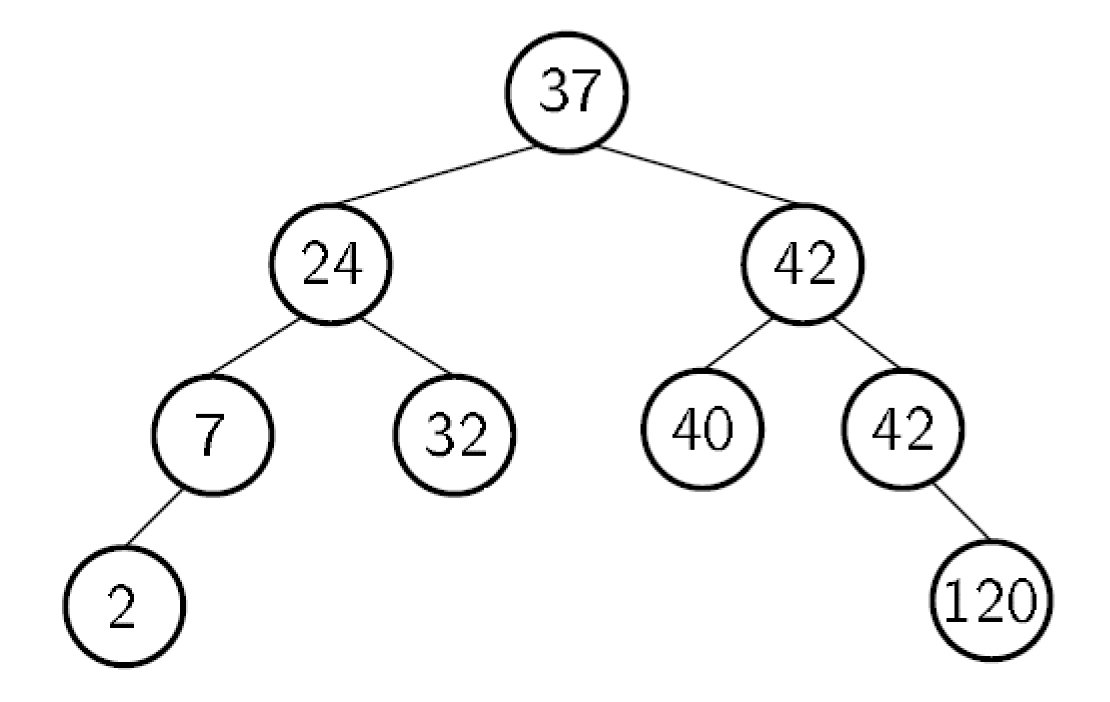
- Buscar

```
public NodoBinario buscar(TipoDeDato valor) {
    return privBuscar(raiz, valor);
}

private privBuscar(NodoBinario nodo, TipoDeDato valor) {
    si (nodo == NULL) retornar NULL;
    sino si (valor == nodo.dato) entonces retornar nodo;
    sino si (valor < nodo.dato) entonces
         retornar privBuscar(nodo.izquierda, valor);
    sino retornar privBuscar(nodo.derecha, valor);
}
```

Operaciones elementales

- Buscar el 32
- Buscar el 40
- Buscar el 120



Operaciones elementales

- Insertar

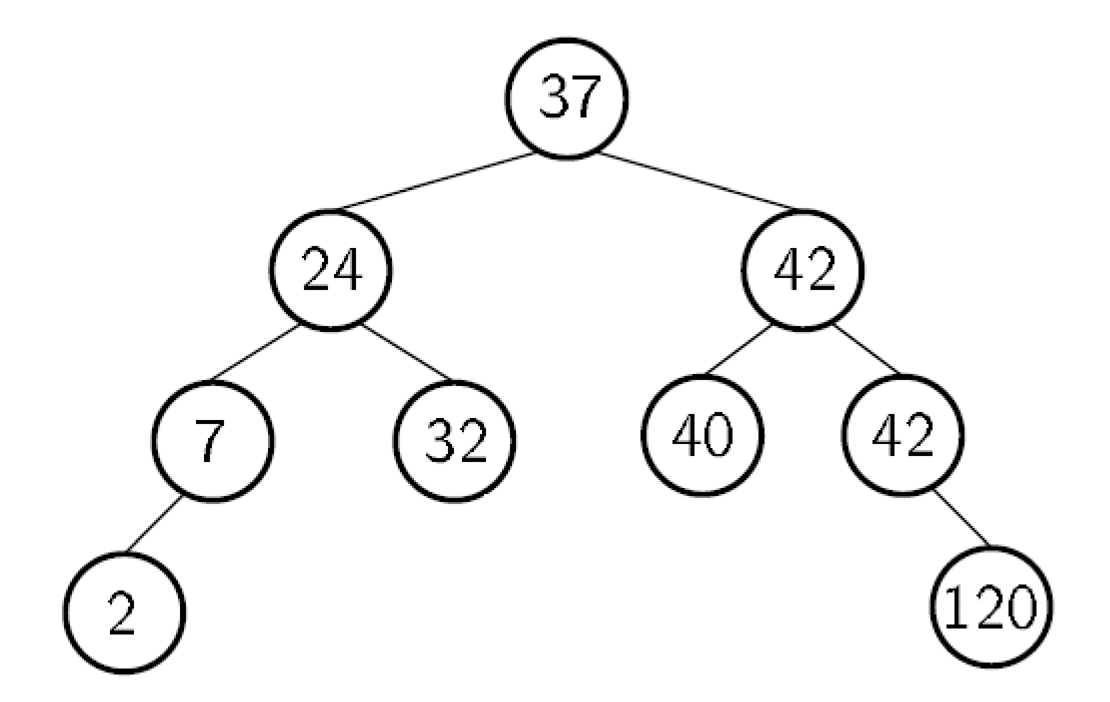
```
public void insertar(TipoDeDato valor) {
    raiz = privInsertar(raiz, valor);
}


NodoBinario privInsertar(NodoBinario nodo,TipoDeDato valor){
    si (nodo == NULL)
         nodo = new NodoBinario(valor, null, null);
    sino si (valor < nodo.dato) entonces
     nodo.izquierda = privInsertar(nodo.izquierda, valor);
    sino
      nodo.derecha = privInsertar(nodo.derecha, valor);

    retornar nodo;
     }
```

Operaciones elementales

- Insertar el 35
- Insertar el 41
- Insertar el 6



Operaciones elementales

- Eliminar

```
public void eliminar(TipoDeDato valor) {
    raiz = privEliminar(raiz, valor);
}
//funcion aux que retorna y borra el min del arbol con raiz nodo
private NodoBinario borrarMinimo(NodoBinario nodo) {
    si (nodo.izquierda == NULL)
         retornar nodo.derecha;
    sino
         nodo.izquierda = borrarMinimo(nodo.izquierda);
         retornar nodo;
}
```

Operaciones elementales

- Eliminar

```
//funcion aux que retorna el min del arbol con raiz nodo
private TipoDeDato getMinimo(NodoBinario nodo) {
    si (nodo.izquierda == NULL)
        retornar nodo.dato;
    sino
        retornar getMinimo(nodo.izquierda);
}
```

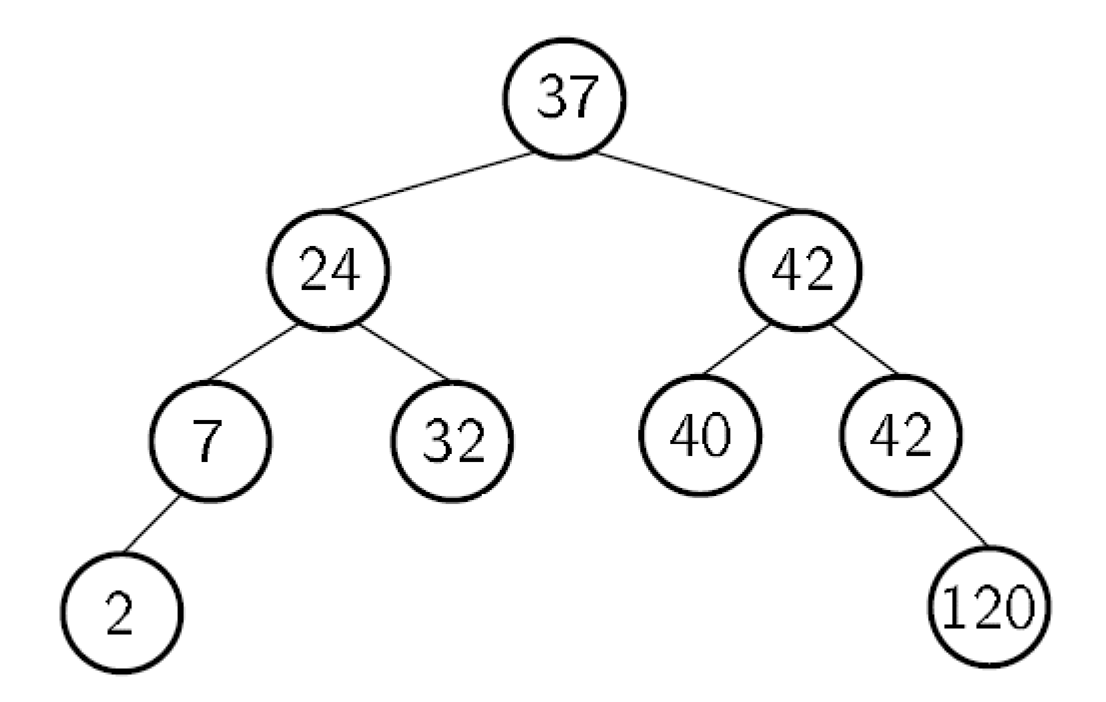
Operaciones elementales

- Eliminar

```
private privEliminar(NodoBinario nodo, TipoDeDato valor) {
    si (nodo == NULL) retornar NULL;
    sino si (valor < nodo.dato) entonces
      nodo.izquierda = privEliminar(nodo.izquierda,valor);
    sino si (valor > nodo.dato) entonces
      nodo.derecha = privEliminar(nodo.derecha, valor);
    sino //encontramos el nodo a eliminar
      si (nodo.izquierda == null) nodo = nodo.derecha;
      sino si (nodo.derecha == null) nodo = nodo.izquierda
      sino //tiene 2 hijos
         nodo.dato = getMinimo(nodo.derecha);
         nodo.derecha=borrarMinimo(nodo.derecha);
         retornar nodo; }
```

Operaciones elementales

- Eliminar el 120
- Eliminar el 24
- Eliminar la raiz



Gracias por su Atención

¿Consultas?

Referencias

- Estructuras de Datos en Java. Mark Allen Weiss,Capítulo 17 y 18.
- A Practical Introduction to Data Structures and Algorithm Analysis. Clifford Shaffer, Cap. 5 y 6.
- ECE 250 Data Structures and Algorithms, University of Waterloo. *Douglas Wilhelm Harder*